VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Technologie vývoje her na platformě Android

Game Development on Android Platform

2013

Martin Škorec

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Martin Škorec**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Technologie vývoje her na platformě Android
Game Development on Android Platform

Zásady pro vypracování:

Technologie vývoje her zažívá v posledních letech velký vzestup. Zejména v mobilních zařízeních můžeme pozorovat obrovský nárůst zájmu uživatelů. Díky tomu se tato oblast vývoje stává rok od roku atraktivnější jak pro jednotlivé vývojáře, tak i pro samotné firmy, které se specializují na vývoj herních technologií pro mobilní zařízení. Cílem práce je vytvořit herní aplikaci na platformě Android.

1. Seznamte se s platformou Android a její architekturou.
2. V popisu se zaměřte především na informace týkající se vývoji her na této platformě.
3. Popište její výhody i nevýhody s porovnání s ostatními mobilními platformami.
4. Po nastudování a popisu technologie vývoje, promyslete výběr herní aplikace a tuto vybranou hru implementujte.
5. V experimentální části popište použité techniky (např. herní engine, herní logiku, návrh umělé inteligence nepřátel, detekce kolizí, simulace pohybu).
6. Zhodnoťte nároky vytvořené aplikace na HW vybavení mobilního telefonu.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Radovan Fusek**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 6. května 2013 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 6. května 2013 .....

Rád bych na tomto místě poděkoval panu Ing. Michalovi Krumniklovi, za úvod a seznámení s platformou Android a panu Ing. Radovanu Fuskovi, za vedení a rady při psaní této práce.

Abstrakt

Tato práce se zaměřuje na vývoj her na platformu Android. Téma je dnes velmi zajímavé, z důvodu rostoucí popularity mobilních zařízení, kde je tato platforma v dnešní době právě nejrozšířenější. Vývoj videoher sebou nese mnohá úskalí, jenž budou nastíněna a analyzována v různých kapitolách. Jedna z her – Battle City, pak bude implemntovana zcela od základu, kdy nám budou dostupné pouze grafické zdroje. Pokusím se také nastínit řešení umělé inteligence bojující proti uživateli. V práci se podíváme, jaké typy videoher a jaký koncept, z pohledu typu zábavy a herního scénáře přitahuje uživatele nejvíce. Závěrem pak zhodnotím nároky na aplikaci i možné rozšíření, nebo alternativy, které by naší hru učinily ještě zajímavější.

Klíčová slova: Android, Vývoj her, Umělá inteligence

Abstract

This Thesis is aiming to a game developing on the Android platform. Topic is very interesting nowdays, because of a growing popularity of a mobile devices, where is exactly this platform mostly common in these days. Game Developing carry many obstacles, which will be sketch and analaysis in a various chapters. One of the games – Battle City, will be immplemented from the scratch, where we will have a graphics resources only. I also try to show a solutions of the artifical inteligence, fighting against the user. In this Thesis we take a look, which kind of the videogames, and which concept is most popular for users, from entertainment and game scenario point of view. In the end I evaulate requirements of this application as well as a possible extensions, or alternatives that will make our game even better.

Keywords: Android, Game development, Artificial Intelligence

Seznam použitých zkratek a symbolů

AI	–	Artificial Intelligence
AOS	–	Android Operation System
DVM	–	Dalvik Virtual Machine
GITTA	–	Geographic Information Technology Training Alliance
iOS	–	iPhone Operation System
JavaSE	–	Java - Standart Edition
JIT	–	Just In Time - compiler
JNI	–	Java Native Interface
LCD	–	Liquid Crystal Display
NDK	–	Native Development Kit
OS	–	Operation System
RAM	–	Random Access Memory
SQL	–	Structured Query Language
SDK	–	Standart Development Kit
VM	–	Virtual Machine
XML	–	Extensible Markup Language
ZTE	–	Zhong Xing Telecommunication Equipment

Obsah

1 Úvod **5**

2 Popis vybrané hry **6**

3 Android a jeho architektura **7**
3.1 Historie 7
3.2 Architektura 7
3.3 Systém a vývoj 9

4 Vývoj her **11**
4.1 Historie 11
4.2 Současnost 11
4.3 Všeobecné know-how 11
4.4 3D vs. 2D 12

5 Android vs. ostatní platformy **13**
5.1 Technické detaily a market 13
5.2 Licence 13
5.3 Dalvik vs. rekurze 14

6 Implementace **15**
6.1 Architektura 15
6.2 Umělá inteligence 15
6.3 Fyzika 16
6.4 Kolize 17
6.5 Grafika 17

7 Vývojová část - Battle city **19**
7.1 Všeobecný popis 19
7.2 Poměry jednotek ve hře 19
7.3 Hrací pole 23
7.4 Jednotlivá kola (prostředí) 23
7.5 Kolize 24
7.6 Modely a AI 24
7.7 Simulace pohybu a vykreslování 26

8 Závěr **30**

9 Reference **32**

Seznam tabulek

1 Naměřené časy 30

Seznam obrázků

1 Screanshot z vybrané hry 6
2 Zásobníkově orientovaná VM 8
3 Registrově orientovaná VM 9
4 Screen shot – Jednotlivá pole logické dělení 12
5 Malířův algoritmus – V tomto případě by algoritmus selhal a zacyklil se . 18
6 Sprite – ukázka 18
7 Nejvyšší vrstva - Dalvik vs Java – schéma 20
8 Nejvyšší vrstva – třídní diagram 21
9 Herní vrstva – třídní diagram 21
10 Běh aplikace – sekvenční diagram 22
11 Ukázka postupu AI jednotek – scénář I 25
12 Ukázka postupu AI jednotek – scénář II 26
13 Sprite – Třídní diagram 28
14 Sprite – Fázový diagram 29

Seznam výpisů zdrojového kódu

1 Pseudokód Dijikstrova algoritmu 15
2 XML schéma 24

1 Úvod

Vývoj a nárůst počtu mobilních zařízení, zejména chytrých telefonů, v poslední době stoupá raketovou rychlostí. V roce 2012 bylo prodáno 491 miliónů zařízení a z toho přes 60 % s operačním systémem Android, druhý zůstává Apple s přibližně 18 % [1]. Stejně jako u stolních počítačů, je hlavním motivem pro růst výkonu grafických karet a procesorů výkon softwaru. S narůstajícími nároky na software přirozeně stoupají nároky na hardware. Samostatným odvětvím jsou herní aplikace, které tlačí především jak na paměť (RAM), tak na výkon procesorů – grafických i řídicích. Nejprve se seznámíme s vybranou aplikací a po té s Android architekturou. V práci budou analyzovány všechny vrstvy tohoto operačního systému a pokusím se nastínit koncept, na kterém je postaven. V druhé části, proberu vývoj herních aplikací pro platformu Android z pohledu uživatele, a toho jaké typy her v současnosti na mobilní zařízení nejvíce uživatelskou komunitu přitahují. Plynule pak přejdeme k srovnání s ostatními platformami a vysvětlíme si licence a filozofii trhu s aplikacemi. Právě způsob jakým společnosti trh řídí a kontrolují je jeden z klíčových aspektů při výběru platformy. V další části popíši problematiky týkající se návrhu a zpracování herní aplikace, jako je fyzika 6.3, vykreslování 6.5, či základy programování umělé inteligence 6.2. Jak se dozvíme problematika videoher je velmi obsáhlá proto tyto kapitoly spíše nastíní možná řešení, než aby je popisovala do hloubky. Ve vývojové fázi 7 budete mít možnost nahlédnout do konkrétního řešení jednotlivých bodů z kapitoly 6. Ukáži, jak je engine postaven, či jak byl řešeno vykreslování, kolize, nebo umělá inteligence. Srovnám také výkon jednotlivých zařízení, na kterých byla aplikace testována a nároky na přístroj samotný. V závěru se pak pokusím shrnout budoucí možná rozšíření a podíváme se na přehled výsledků, kterých jsem v práci dosáhl.

2 Popis vybrané hry

K prozkoumání vývoje herních aplikací na platformě Android jsem zvolil předělanou verzi známé hry Battle-City na Nintendo© 8-Bitovou herní konzoli. Tato hra je typu 2D a lze v ní najít níže popsanou funkcionalitu. AI, kolize a paralelní běh více vláken. Hra obsahuje nepřátelské entity, grafické artefakty, jenž se v průběhu hry dají ničit, a také rozmanité kola.

Nastíním spousty netriviálních problémů a popíši různé techniky jak je elegantně vyřešit. Místo tvoření nové hry s novým příběhem i myšlenkou, jsem tedy zvolil již navrženou hru. Můžeme tak srovnat, jak blízkého výsledku jsme schopni dosáhnout a ušetří se čas na návrh hry i designu. Ten bude místo toho vynaložen na vývoj kompletního herního enginu. Zvláště velkou výhodou je možnost stažení grafických modelů ze hry přímo z internetu. Zde je obrázek z původní hry:

Obrázek 1: Screanshot z vybrané hry

Herní scénář je následující: Hráč má za úkol chránit orlici situovanou v dolní části hracího pole a zlikvidovat všechny nepřátelské jednotky. Jejich počet je konečný stejně jako to, kolik se jich může maximálně najednou ve hře objevit. Po vyhraném kole hráč postupuje do dalšího. Hra končí v momentě kdy AI (Artifical Inteligence – Umělá Inteligence) jednotky zničí orlici nebo hráč nemá další životy. Ty se doplňují pomocí bonusů, které můžou obsahovat i jiné zvýhodnění - jako dočasnou nesmrtelnost, instantní zničení všech AI entit v poli, atd. V původní verzi byla AI jednoduchá a založená spíše na náhodném pohybu. V naší verzi má k dispozici sofistikovanou metodu jak vypátrat - zneškodnit hned na začátku. Hráči ale usnadňujeme první kola jinými způsoby, jenž popíši později. Nepřátelé mají 4 verze - pancéřovou (vydrží 2 zásahy), rychlou, s dvoj-střelbou a standartní bez zvýhodnění.

Samozřejmě přidáme grafické prvky jako animace, zvuky či ukládání výsledků, které ale popíši pouze okrajově.

3 Android a jeho architektura

3.1 Historie

Zmínka o Androidu (Android Operation Systém - AOS) se poprvé objevuje v roce 2005 založením malé firmy Android Inc. Tím nastaly spekulace, že Google hodlá vstoupit na trh s mobilními telefony. O pár let dříve, totiž nastal raketový vzestup smarth phonů a Google nechce zaspat svou šanci. V roce 2008 vychází Android 1.0 a stává se novým konkurentem na trhu mobilních operačních systémů. Konkurenti mu jsou iPhone OS (iOS) a BlackBerry[5].

Od té doby prošel AOS zatím sedmi updaty všemi pojmenovanými podle dezertů, každý pak dodával další funkcionalitu. Od verze 1.5 je možné využívat nativních funkcí (psaných v C++ a C) – před tím bylo možno psát aplikace čistě v Jave. Verze 1.6 přináší možnost podpory různých rozlišení obrazovky, od verze 2.0 přichází multi-touch podpora a verze 2.2 přináší důležitý JIT kompiler popsaný níže (pro více informací viz. [5] .

Nutné dodat, že AOS je kompletně open-source (možnost náhledu do zdrojových kódů a jejich modifikace není trestně stíhána). To je obrovská výhoda a možná právě toto rozhodnutí Google učinilo z AOS jeden z nejoblíbenějších a nejrozšířenějších operačních systému pro mobilní zařízení na světě. V současnosti se odhaduje, že na Android marketu je asi 700 000 aplikací [2]. Open source přístup zajistil vývoj pro všechny cenové relace mobilních zařízení. Je pak na výrobci, aby dodal drivery a umožnil běh tohoto systému na svém výrobku a přispůsobil AOS technickým nárokům zařízení. V současnosti se také například uvažuje i o jeho možném rozšíření do auto-rádií.

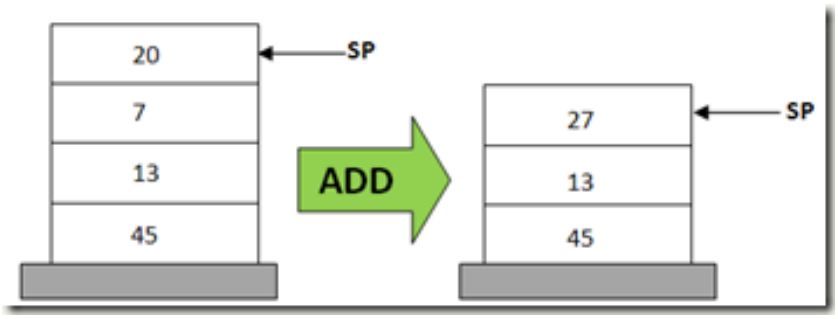
3.2 Architektura

3.2.1 Kernel

Android je platforma postavená na linuxovém kernelu. Byla modifikována pro potřeby mobilních zařízení Googlem ale od Linuxového jádra se příliš neliší. Byl vybrán z důvodu snadné přenositelnosti mezi různými zařízeními. Obsahuje drivery, které jsou dodávány výrobcem. Je tak nejnižší vrstvou mezi fyzickým zařízením a softwarem a je zodpovědná také za kontrolu spotřeby, řízení paměti a podporu jejího sdílení a o sítovou podporu.

3.2.2 Nativní knihovny

Další vrstvou jsou C++/C nativní knihovny dodávané Androidem (Googlem). Knihovny obsahuji funkce pro práci s různými formáty medií, engine pro vnitřní SQL databázi nebo například surface manažera zajištující nepřímé vykreslování (Off-screen buffering). Místo aby se kreslilo přímo na obrazovku, vykresluje se do bufferu, kde se jednotlivé vykreslení kombinuje s dalšími a teprve po té je zobrazeno na display. Dále NDK obsahuje grafické knihovny jako OpenGl a SGL (Skia Graphics Engine).



Obrázek 2: Zásobníkově orientovaná VM

3.2.3 Dalvik

Dalvik Virtual Machine (Dalvikův Virtuální Stroj) vznikl pro účely Androidu a měl za úkol brát požadavky na malou pamět, spotřebu a rychlost cílových zařízení. Ve srovnání s klasickou Javou je asi 2-3 pomalejší. Od androidu 2.2 má navíc DVM k dispozici vlastní JIT kompilátor. Je navržen tak, aby byl možný (a hlavně efektivní) souběh vícero instancí DVM.

3.2.4 DVM vs JVM

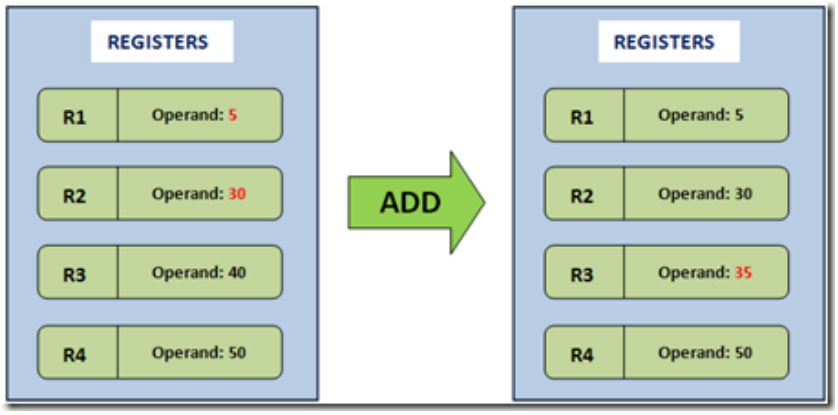
Systém Poslední nejvyšší vrstvou je DVM (Dalvik Virtual Machine). Dalvik je často srovnáván s Javou(Java Virtual Machine) nebo i mylně zaměňován. Na rozdíl od zásobníkově Javy je Dalvik registrově orientovaný.

U zásobníkově orientovaného stroje tzv. LIFO (Last-In Fisrt-Out), je každý operand uložen do zásobníku jak ukazuje obrázek **??**. Výhodou je že si není potřeba pamatovat adresu operandu – hodnoty. Je adresovaná implicitně ukazatelem na vrchol zásobníku. Ovykle se preferuje tato architektura prevalentní u většiny realných procesorů. Výhodou je simplicita kompileru kódu pramenící z faktu, že instrukce jsou krátké a mají přirozené adresování. Odpadá tedy nutnost režie na úkládání hodnot často prováděných operací a není nutná tak velká optimalizace.

U registrově orientovaného stroje (obrázek 3) , se explicitně odkazuje na místo v paměti, kde je operand uložen. Výhodou zásobníku je mnohem kratší výsledný kód jelikož není nutné specifikovat adresy operandů (stačí pouze příkazy pop, push). Výhodou registrů je pak možnost mezi-uložení proměnné pro pozdější účely a menší režie - odpadají neustále pop a push operace a tudíž i méně instrukcí. [3].

Java pak využívá 8-bitovou instrukční sadu. Dalvik má vlastní 16-bitovou sadu a pro registr operandů používá 4-bitové pole. Z dalvika je také možné volat C/C++ funkce pomocí JNI technologie překompilované pomocí NDK. Výhodou je rychlejší běh větší možnosti (hlavně ve využití paměti) a přenositelnost knihoven v tomto jazyce. Nevýhodou je těžkopádnost lazení chyb a ne všechna zařízení NDK podporují.[3].

Byte kód



Obrázek 3: Registrově orientovaná VM

Zdrojový kód pro Dalvik je psaný v Jave a je překompilován do JavaByte kódu (.class soubor) a po té znovu překompilován do .dex souboru obsahující bytecode pro Dalivka. Syntaxe je proto stejná jako v Jave. Nevyužívá však všechny Java knihovny, ale místo toho využívá některé vlastní založené na Apache Harmony Java implementaci.

3.3 Systém a vývoj

3.3.1 Spouštění a běh

Spouštění probíhá v těchto krocích:

- Po nabootování systému, boot nahrávač nahrává Kernel do paměti.
- Kernel spouští Init program, který je předkem pro všechny další procesy v systému.
- Init vytváří systémové vlákna a hlavně Zygote servisu.
- Zygote servis je zodpovědný za rychlé vytváření nových instancí DVM při příchozím požadavku a za sdílení dat a objektů mezi aplikacemi.

Každá aplikace má k dispozici vlastní instanci DVM, čímž se vyvaruje hromadného pádu aplikací. Mezi aplikacemi je možné snadno sdílet data z databází aplikací (pomocí ContentProvider rozhraní) spouštět jiné procesy.[3].

3.3.2 SDK

SDK-Software Development Kit obsahuje balíček nástrojů, dokumentace , tutoriálů a příkladů, jenž začínajícímu programátorovi umožní vyvíjet plnohodnotné aplikace ve velmi krátkém čase. Mezi hlavní nástroje, které SDK poskytuje patří:

- Debugger skrz nějž je možné aplikaci ladit. Jak na emulátoru v PC tak na reálném zařízení.

- Paměťový a výkonový profil sloužící k měření zátěže na zařízení a odhalení přečerpání paměti a pomalých částí kódu.
- Emulátor - simuluje reálné zařízení v PC. Pro jeho pomalost se ale využívá spíše k návrhu uživatelského rozhraní než k lazení funkcionality.
- Příkazovou řádku pro komunikaci se zařízením (možnost rootu zařízení)
- A samozřejmě build skripty pro sestavení a nasazení aplikace na zařízení.

Přiloženy jsou také Java knihovny nezbytné pro vytvoření Android aplikací, protože jak už víme Dalvik je derivát Javy. API – knihovny pro jednotlivé verze androidu si pak vývojář stáhne podle toho na jakou verzi má zájem aplikaci tvořit. SDK je pak integrováno do Eclipse, populárního vývojářského a na funkcionalitu bohatého prostředí. Integrace je možná také do NetBeans, kde ale chybí oficiální podpora.[5].

3.3.3 Prostředí pro vývoj Her

Android podporuje již zmiňovanou OpenGl knihovnu. Ta, i když v nižších verzích než na PC, obsahuje dostatečnou funkcionalitu a rychlost pro vývoj jednoduchých 3D her. Většinou se jedná o 3D plošinové hry, či předělané a upravené verze PC her pro mobilní zařízení. I bez OpenGl je možné vytvořit hru čistě pomocí Javy, bez použití 3D grafických funkcí se bude jednat jen o 2D hru. K ovládání má nyní Android dostatečné množství gyroskopických a dotykových rozhraní, přes které se lze vytvořit ergonomicky přijatelné ovládání. I tak u 3D her na mobilní zařízení myš a klávesnice značně chybí.

4 Vývoj her

Když si chce člověk skutečně hrát, musí si hrát jako dítě.
-Johan Huizinga

4.1 Historie

Vývoj mobilních videoher začal již s prvními verzemi víceřádkových – zatím pouze monochromatických – LCD displeyů. Jedna z nejznámějších videoher Snake na tehdy rozšířenou značku Nokia vyšla již v roce 1997. S rozšířením barevných displejů někdy okolo roku 2002 dostávají videohry na mobilních platformách jiný rozměr. Zhruba ve stejnou dobu se objevují také první smartphony a s nimi také první pokročilejší videohry.

4.2 Současnost

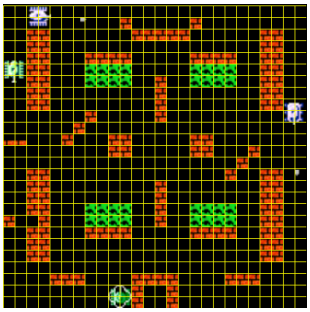
Vývoj her na mobilní zařízení v posledních letech získává nový punc. S narůstajícím výkonem procesorů jde na mobilních platformách vytvářet i zdařilé 3D hry. Jde také o ovládání videoher, které je na mobilním zařízení značně omezené a je třeba hledat způsob jak uživateli ovládání co nejlépe usnadnit. Nahrává tedy spíše arkádovému a na ovládání jednoduchému stylu. Popularitu tak stále mají i jednoduché 2D videohry kdy nejde vždy o propracované a nejpokročilejší grafické a vizuální efekty, ale o jednoduchou a pokud možno návykovou zábavu. Například 2D hra Angry Birds pro platformu Android slavila v roce 2012 miliardu stažení [4]. Sází přitom na flashový kreslený design, jednoduchou fyziku a zábavnou pointu hry. Na trhu je také značné množství 3D titulů. Opět se ale jedná o simplicitní a na ovládání intuitivní zábavu.

Závěrem si musíme uvědomit, že mobilní telefony, tablety či jiné kapesní počítače sice dnes mají technické parametry blížící se více a více stolním počítačů. Stále jsou to ale komunikační a pragmaticky zaměřená zařízení, která umožnují také hraní náročnějších videoher. V současnosti je ale největší překážkou kapacita baterií – graficky náročnější hru si tak uživatel příliš dlouho bez nabíjecího zařízení nezahraje.

4.3 Všeobecné know-how

Vývoj her je náročný. Ne kvůli náročnosti na kód samotný oproti např. průmyslovým simulacím nebo leteckému softwaru, ale z důvodu obrovského množství informací a dílčích postupů, které je nutné nastudovat. Vezměme v úvahu návrh samotné architektury, a přenesitelnost na jiné operační systémy, nebo do dalších programovacích jazyků. Definici fyzikálních zákonů a kolizí, či způsob jakým svět ve hře zobrazit a jak jej animovat, nebo po té následně jak hru zastavit pokud to hráč vyžaduje. Neméně důležitou věcí je pak scénář samotné hry, či její myšlenka a zaměření. Spousty her nespatřilo světlo světa jen z důvodu, že ztroskotaly v této fázi, jak se blíže píše v [5].

Jak vidíme je zde mnoho aspektů, ty musí být promyšleny nejlépe v začátcích vývoje, jelikož později jsme pak nuceni tyto problémy řešit ad hoc. To má za následek nepřehled-



Obrázek 4: Screen shot – Jednotlivá pole logické dělení

nost kódu a mnohem více práce s předěláváním systému. Pak i na první pohled triviální změna, může znamenat modifikaci velké části kódu, což je jistě časově velmi náročné.

U vývoje videoher neexistují jednoznačně a striktně daná pravidla, pouze určitá paradigmata a všeobecně doporučené postupy. Na rozdíl třeba od vývoje informačních systému, kde existuje určitá konvence, či zaběhnuté principy.

4.4 3D vs. 2D

Vývoj a implemtace 3D her je oproti 2D hrám časově enormně náročná a je třeba kvalitního enginu, propracovaných 3D modelů i dobré herní logiky. Pohyb či inteligence nepřátel se rovněž rozpadá do mnoha složitějších úloh (robot path planning např.). U her se doporučuje využít herní vlákno asynchronně běžící od UI a hýbající předměty. Modely by pak měly umět detekovat kolize, nebo poskytovat obaly pro jejich výpočet. Návrhy implementace nepřátel mohou být založené na náhodném pohybu, nebo pohybu průsečíkem k hráči přičemž při nárazu do překážky je objekt otočen a vyslán jinou cestou. Také u některých návrhů je herní plocha rozdělena na grid, kdy jednotlivé čtverce představují pole, kam se jednotky přesouvají.

Oproti 2D hrám jsou detekce kolizí náročnější využívajíce buď to vlastních, nebo frameworkových herních engínů. Těch je na trhu spousty jak pro 2D tak pro 3D videohry. Psaní vlastního engínu je zdlouhavé a obnáší vyrovnání se, se všemi výše uvedenými problematikami.

5 Android vs. ostatní platformy

5.1 Technické detaily a market

Android na rozdíl od JavaME nemá build-in třídy na animaci pohybů a vykreslení map. Je tedy nutná vlastní implementace. Zatímco AOS a konkurenční IOS sází na opensource OpenGL knihovny a to speciální odlehčené 2.0 ES verze, Windows Mobile podporuje DirectX (více informací 9[6] [7]). Je ovšem omezeno oproti PC využitím paměti.

Dle mého názoru vývoj her na mobilní zařízení a vybrání vhodné platformy je čistě otázkou marketingu. Výkonově je, co se her týče, nejlepší platformou Andoid. Z pohledu marketingu zase Apple pro jeho filozofii trhu s aplikacemi. Je mnohem těžší stáhnout hru z App Store na černo, na druhou stranu Android dává příležitost mnohem větší komunitě vývojářů. Je obtížné říci, zda je přístup kvantity – z pohledu Androidu, nebo kvality - z pohledu Apple správný. Vývojář i uživatel bude apelovat na volný přístup, či širokou komunitu s nadšenci plnící fóra. Manažeři a majitelé společností vyvíjející software, zase na konkurenční boj a rentabilitu. Přístup Microsoft - Windows Mobile spoléhá na propojení chytrých telefonů s Xbox konzolí. Všeobecně je ale vnímán jako platforma pro uživatele, který od telefonu příliš širokou škálu herních a doplňkových aplikací nevyžadují a své zařízení vnímají spíše z pragmatického hlediska.

5.2 Licence

Neméně důležité jsou i licence. Před tím než vývoj na danou platformu začne je třeba zajistit licenční práva, nakoupit reálná zařízení a také výpočetní techniku schopnou software sestavit a nasadit. Android má v tomto směru poměrně velkou výhodu. SDK je kompletně zdarma a dostupný pro Windows 32-64 bit, Linux 32-64 bit a Mac OS X 32-64-bit. Pro Mac OS není dostupný ADT (emulátor). Pro vývoj aplikací na Apple je nutné mít nainstalovaný Mac OS, vývoj na Windows OS, nebo Linux není oficiálně podporován a existují pouze neoficiální způsoby jak toto omezení obejít.

Aplikace na Microsoft i na App Store prochází přísným schvalováním a ručí tak za absenci škodlivého kódu. Je zde také více kontrolována funkčnost, čímž se filtrují chyby obsahující, nebo amatérský software (čerpáno z [8] [9]). Android má odlišný přístup - na jejich Market smí nahrávat aplikace jen ti developeři, kteří si vygenerují privátní klíč. Tím pak svou aplikaci podepíší a následně nasadí na trh [10]. Google tak sází na zcela otevřený a bezplatný přístup na svůj trh. Verifikace aplikací je od verze 4.2 ponechána na uživatelích tím, že si aplikace instalované v zařízení zanalyzuje sám Android. [11]. Buď průběžně, nebo ještě před samotným stažením. Po té vyhodnotí, zda-li aplikace může být škodlivá a zároveň odešle data na server, kde je vedená databáze takto označených softwarů. Na Android například, lze také stahovat aplikace z neznámých zdrojů. Což je na iPhone možné pouze při změnách, po kterých ale uživatel přichází o záruku.

5.3 Dalvik vs. rekurze

Během vývoje bylo také zjištěno, že původní návrh jedné z hlavních metod ve hře není možné na Androidu implementovat. Při velké rekurzi docházelo k přetečení zásobníku. Zkoumáním různých zdrojů bylo zjištěno, že u interpetovaných jazyků (Dalvik) iterace navíc může porážet hlubokou rekurzi, pokud není implementována jako koncová, kterou za určitých okolností překladač rozpozná.

6 Implementace

Kdo ví proč, dokáže jakékoli jak.
-Friedrich Nietzsche

6.1 Architektura

Architekturu je třeba promyslet. Hra by měla běžet ve více vláknech. Vlákno si lze představit jako hnací jednotku, která daný kód vykonává. U her se využívá uzavřených smyček, ty se opakují tak dlouho, dokud nedojde k ukončení hry. Vlákna pak v cyklu provádí jednotlivé operace dokola. Koncepčně se doporučuje používat modely pro hráčovy jednotky, které nesou informaci o stavu, ve kterém se nachází o pozici a nesou další doplňková data důležitá pro hru. Podrobnněji se touto problematikou budeme zabývat v kapitole 7.6. Engíne pak na základě hráčovy interakce do herního scénáře zasahuje. Používá se mnoho pomocných tříd a návrhových vzorů. Např. inicializace nových jednotek se může provádět za pomocí továrních metod. Pokud je k vytvoření nové jednotky třeba mnoho parametrů, může je nastavovat pomocná třída. Té se předají pouze úzce specifické údaje, na základě kterých si třída ostatní parametry buďto sama zjistí, nebo dopočítá. Dalším aspektem je problém s přístupem k sdíleným proměnným. Běží-li aplikace ve více procesech pracuje daleko rychleji, ale je nutné zajistit, aby měl k určitým proměnným přístup v danou dobu pouze jeden proces. Je nutná synchronizace procesů použitím monitorovacích zámků (v Jave klíčovým slovem synchronized), nebo semaforů, čímž zajistíme, že k dané proměnné má přístup pouze jeden proces – vlákno. Problematika se týká třeba kolekci vykreslovaných artefaktů. Jeden proces nesmí tuto kolekci mazat a jiný zároveň vykreslovat

6.2 Umělá inteligence

Dijkstra

K vyhledávání cesty můžou sloužit různé algoritmy. Např Dijkstrův byl koncipován již v roce 1956 Edsgerem Dijkstrem a slouží k vyhledávání nejkratší cesty grafem.

Prekondice: Všechny uzly mají nejlepší vzdálenost nastavenu na "nekonečno"což je vzdálenost, která bude vždy větší, než vzdálenost z kteréhokoliv startovního bodu. Všechny uzly jsou také nenavštívené, nemají referenci na předchozí uzel a nemají nejlepšího souseda Zde je pseudokód 1 zdroj : [16].

```
1:  function Dijkstra(Graph, source):
2:  for each vertex v in Graph:  // Inicializace
3:  dist[v] := infinity  // Pocatecni vzdalenost start − cil je nastavena na nekonecno
4:  previous[v] := undefined  // Predchozi uzel je optimalni cesta k cili
5:  dist[source] := 0  // Vzdalenost start−start
6:  Q := the set of all nodes in Graph  // Vsechny uzly v Q jsou neoptimalizovane
7:  while Q is not empty:  // Hlavni smycka
8:  u := node in Q with smallest dist[ ]
9:  remove u from Q
10: for each neighbor v of u:  // Kde v jeste nebylo vyjmuto z Q
11: alt := dist[u] + dist_between(u, v)
```

```
12:   if alt < dist[v]   // Relax (u,v)
13:   dist[v] := alt
14:   previous[v] := u
15:   return previous[ ]
```

Výpis 1: Pseudokód Dijikstrova algoritmu

Algoritmus tedy vykonává práci tak dlouho, dokud nejsou všechny uzly výjmuty z množiny Q. Algoritmus později implementovaný ve hře z Dijkstrova algoritmu vychází, ale pro účely hry byl modifikován.

A* (”A star”)

Jde o algoritmus fungující na podobném principu jako Dijkstrův. Tento ale používá k určení nejkratší cesty grafem Heuristickou metodiku. K uražené vzdálenosti při průchodu grafem může například přičítat reálnou vzdálenost k cíli. Pokud by vzdálenost rostla, dal by se jinou cestou. Zvlášť výhodný je ve chvíli kdy nás průchod celým grafem nezajímá, nebo v případě kdy je dobré si nějakou cestu zapamatovat. Místo reálné vzdálenosti můžeme uzlům určit jak výhodná cesta je, nebo z pohledu opakujících se událostí, jak výhodný pohyb na danou pozici může do budoucna být. Dá se tak vytvořit úsudek, kudy by byla trasa grafem asi nejkratší. A právě úsudek je vlastnost, která strojům přirozeně chybí. Slovem graf, nebo cesta grafem, pak můžeme myslet také všeobecné řešení problému pomocí sekvenčně daných kroků (šachy, Rubikova kostka, Loydova patnáctka, atd.).

6.3 Fyzika

Hry jsou velmi dobré v šizení věcí a nejinak je tomu u fyziky. Například při vývoji hry Mafie dali programátorovi za úkol naimplementovat fyziku, která ovlivní pád krychle (krabice) na zem. Údajně přišel s řešením, jenž bylo z pohledu fyziky zcela korektní, ovšem časově náročné. Výsledek? Finální řešení spočívalo v otočení krychle o určitý úhel – třeba 45 stupňů – při každém dopadu na zem, či zásahu projektilem. Zde vidíme, že ne vždy požadujeme realistické řešení problému, ale vystačíme si s empirickým, výpočetně nenáročným a přitom na pohled korektním řešením. Zkrátka a zjednodušeně řečeno je potřeba, aby řešení vypadalo „dobře“.

Fyzika ve hrách využívá hlavně vektorů. Vše pak záleží na konkrétní implementaci, kdy můžeme například vyjádřit směrový vektor rychlosti, z něj parametrickou rovnici přímky a při znalosti počátečního bodu A plynule přejít rychlostí $\vec{v}(x, y)$do bodu B. Parametrickou rovnici získáme ze vztahu:

$$\vec{v} = B - A$$

Kterýkoliv bod na přímce p pak dostaneme ze vztahu:

$$X = A + \vec{v} * t, t \in \Re, v \neq 0$$

Parametrické vyjádření hraje také velkou roli při určování kolizí. Z místa A můžeme

kontrolovat, zda přímka nekoliduje s jinými objekty. Ty musí být samozřejmě definovány také parametricky.

Pokud by jsme chtěli v případě nárazu do mantinelu těleso odrazit jiným směrem, opět by jsme pracovali s jeho rychlostním a směrovým vektorem. Nejjednoduchšší je pouze zaměnit znaménko případě nárazu o mantinel ze strany X ve vodorovném směru, nebo Y pokud dojde ke kolizi ve svislém směru. Pokud by ovšem těleso narazilo do šikminy, je třeba napočítat normálu – a podle ní pak spočítat úhel odrazu.

6.4 Kolize

Řešení kolizí se řeší schránkou okolo objektu.

Častěji se využívají konvexní obaly, které mají těleso zaobalit. Tím je myšleno těleso, pro jehož každý vnitřní úhel platí $\alpha < 180°$. Matematicky se pak dá snáze zjistit, zda došlo (nebo dojde) ke kolizi s jiným předmětem. Ideální je používat základní geometrické primitivy jako čtverec/cuboid, trojúhelník, nebo kruh/sféru. Čím více je kladen důraz na přesnost detekce, tím složitější musí být obal tělesa. [12]

Dobrým přístupem je, prvně zjistit zda ke kolizi došlo a po té případně na situaci zareagovat. Kolize můžeme dělit právě podle reakce tím že postačí:

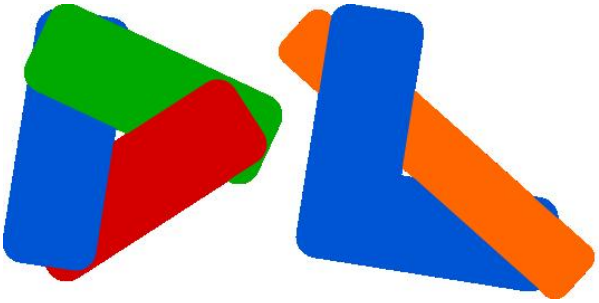
- Pouze potvrzení že ke kolizi došlo a detekce příslušného bodu a normály k tomuto bodu.
- Výpočet minimálního impulzu nutného k odstranění kolize nebo jejímu zabránění. Častá je také změna směru rychlosti.
- Fyzikální změna - deformace předmětu či jeho částí.

V případě reakce na kolizi se využívá tzv. impulzů. Ty mají za úkol reágovat na kolizi tím způsobem, že těleso deformují, změní směr jeho dráhy, nebo rotaci. Při aplikaci impulzu se může zvlašť počítat také s třením, vlastnostmi materiálu, a dalšímy fyzikálnímy vlastnostmi. Vychází přitom hlavně z Newtonových fyzikálních zákonů (více o tomto tématu na [15]). Stejně tak je relativní jak přesná detekce musí být - vše záleží na konkrétní situaci.[12]

Výše popsaná technika v kapitole 6.3 s parametrickou přímkou pak například, může hrát roli při určování cesty AI jednotce. Pokud by naše přímka kolidovala s jiným objektem, definovaným jako čtverec, kruh, nebo jiným geometricky jednoduchým objektem, jsme schopni zjistit přesný bod, kde ke kolizi dojde. Pak už záleží na nás, jak na danou situaci zareagujeme. V tomto případě by nejspíše postačilo zvolit jinou cestu. Z výše uvedených možností tedy potřebujeme pouze potvrzení, jedná-li se o předvýpočet.

6.5 Grafika

Nedílnou součástí her je grafika. Mnohdy je právě ona klíčovým aspektem, kvůli kterého pokračujeme ve hře dál. Našim úkolem je grafiku animovat – oživit, a zajistit její správné vykreslení. Nyní si popíšeme několik algoritmů, které lze využít.



Obrázek 5: Malířův algoritmus – V tomto případě by algoritmus selhal a zacyklil se

Obrázek 6: Sprite – ukázka

Malířův algoritmus – jednotlivé grafické elementy jsou vykreslovány v pořadí podle nejvzdálenějšího po nejbližší. Založen na principu práce malíře, který na podkladovou barevnou vrstvu nanáší další vrstvy postupně od pozadí k popředí scény. Pro každou plochu napočítáme minimální hodnotu ZET souřadnice a podle této hodnoty plochy setřídíme. Může být doplněn i o testy překrývání hran, či ploch. Další možností je rozřezání plocha na menší díly části [13]

Tento algoritmus bohužel selhává v případě, kdy se plochy protínají přes sebe. Jak ukazuje obrázek 5

Z-Buffer – V současnosti asi nejúčinnější algoritmus, nikoliv však nejrealističtější. Pro svou rychlost a snadnou implementaci široce využíván ve videohrách. Základem metody je použití paměti hloubky (z-buffer), která je realizována matici MxxY kde X a Y jsou souřadnice samotných pixelů na obrazovce. Každý element matice obsahuje nejmenší z-hodnoty, které byly doposud zaznamenány. Pokud se má na stejném pixelu vykreslit jiný objekt, je tento pixel kontrolován oproti z-bufferu. V případě, že je hodnota menší pak se pixel obarví barvou nového pixelu (bližšího objektu – plochy).[13]

Animace pomocí spritů (Sprite [sprait] – víla, skřítek) je založená na po sobě jdoucích snímcích. Přehrávají se pak za sebou podobně jako film (viz. obrázek 6). Implementace je poměrně snadná a ve 2D videohrách je to velmi častý způsob animace hlavně u Arkádových videoher a lze ho jednoduše kombinovat s malířovým algoritmem.

7 Vývojová část - Battle city

7.1 Všeobecný popis

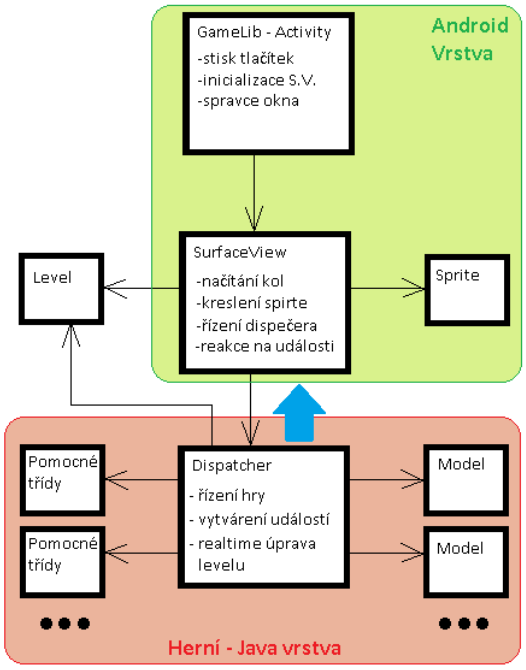
Hra má dvě vrstvy první je plně přenositelná do Javy obsahující herní logiku. Druhá pak psaná pro Android a jsou v ní metody jak vykreslit a zpracovat data ze hry, jak hru řídit a ukládat výsledky. Postup byl zvolen z důvodu potenciální přenositelnosti na jiné Javu podporující platformy (PC, Java-ME). Vykreslení čistě hrací obrazovky, probíhá v třídě SurfaceViewDerivated. Jsou v ní načtené bitmapy a přehrává také zvuky ve hře. Předává také reakce na stisky kláves z nadřazené třídy GameLibActivity a inicializuje herního Dispečera jak ukazuje obrázek 8, který se stará o chod hry tím, že poskytuje své metody vláknům viz. obrázek 9. Dispečer pak udržuje informace o stavu hry a zároveň poskytuje metody pro ovládání její ovládání. V případě, že dojde ke hře k nějaké události, je na to SurfaceViewDerivated upozorněn skrz zpětné volání (callBack) metodou. Jak ukazuje obrázek 7, kde událost je znázorněna modrou šipkou. Místo Android vrstvy by mohla být např. JavaSE. Rozložení tlačítek a odchyt událostí od hlavního (Android UI) vlákna, ukládání stavu hry, nebo vybráním správného kola pak řeší třída GameLibActivity. K Dispečerovy, a ke hře samotné nemá přístup - v hierarchií naší aplikace stojí nejvýše. Komunikuje pouze s uživatelem a přistupuje k uloženým datům. Schéma tak by jsme jej viděli z pohledu Androidu, čili nejvyšší vrstvy demostruje schéma 8.

K běhu aplikace - ta je rozdělena do tří vláken. První dvě, vedená jakožto třídní atribut, kvůli synchronizaci, třetí spouštěno lokálně za účelem překreslení obrazovky 9. Herní vlákno (GameThread) má za úkol řešení kolizí, pohyb modely a jejich interakci a nasazování nových entit v případě, že to stav hry dovoluje. Po kalkulaci kolizí spouští navíc překreslovací, lokální tzn. uvnitř funkce vytvořené vlákno, které překreslí obrazovku viz. fázový diagram 10. Děje se tak, skrze implementovaný interface a události jak je popsáno na obrázku 7. Platí, že jedna iterace hlavního vlákna = jedno herní kolo. Hladký průběh hry zajišťuje uspání herního vlákna na konci každé iterace po určitou dobu. Ulehčuje se tím práce procesoru, vyhlazují rozdíly mezi časově a výpočetně náročnějšími iteracemi a hlavně je rychlost hry ovladatelná. Poslední vlákno PathSetter má za úkol pouze počítání cesty a to za jednu iteraci pro jeden model. Vlákno provádí nejsložitější operace a spí x-krát delší dobu než hlavní herní vlákno. Poměr spánku oproti hlavnímu hernímu vláknu je zadaný konstantou. Stejně rychlost běhu herního vlákna. Vlákna jsou uvnitř Dispečera a vyšší vrstvy k nim už nemají přístup.

Verze tří vláken byla použita, jelikož jinak by herní kolo trvalo průměrně 220-300 milisekund na zařízení s Androidem 2.1. Jak vyplývá z naměřených časů viz. tabulka 1. Hlavním a největším problém byla synchronizace procesů, a přístup k sdíleným proměnným. Hlavní algoritmy a procesy také za svou dobu prošly značnou optimalizací.

7.2 Poměry jednotek ve hře

Obrazovka musí padnout na míru každému zařízení z důvodu vykreslení všech komponent, hlavně pak spodních tlačítek. Je tedy nastaven bezpečný poměr šířky a výšky displeje kdy se komponenty vejdou na displey všechny. Aktuální poměr je počítán jako



Obrázek 7: Nejvyšší vrstva - Dalvik vs Java – schéma

podíl šířky - $width$ ku výšce - $height$. Pokud je aktuální poměr zařízení větší než uvádí bezpečný poměr - $safeRatio$, pak je velikost hrací plochy upravena podle jednoduchého vzorce:

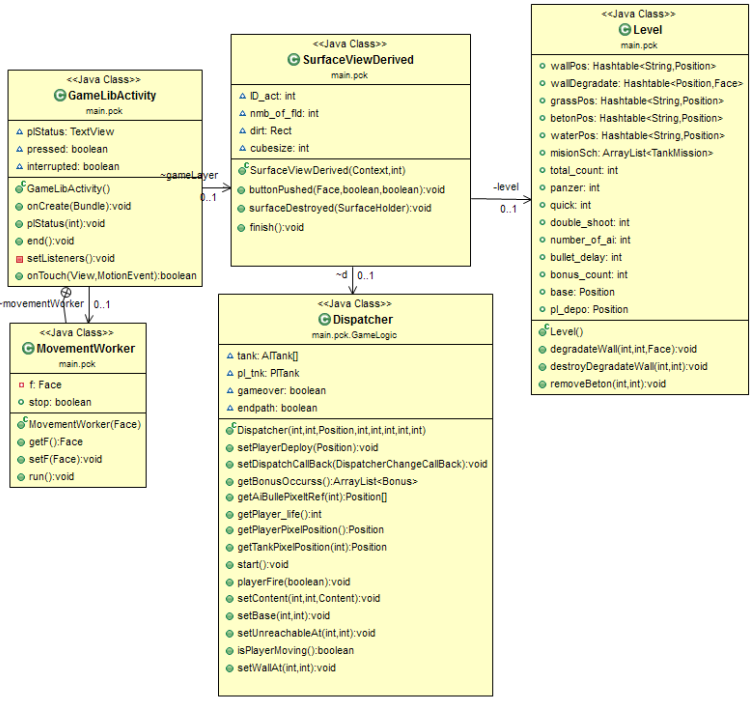
$$width = safeRatio * height$$

Výška je irelevantní, jelikož volný prostor dole může zůstat nevyplněn. V momentě kdy je známo jak široké má SurfaceViewDerivated být, nastaví se velikost jednoho pole. Jeho velikost je zároveň základní jednotkou ve hře.

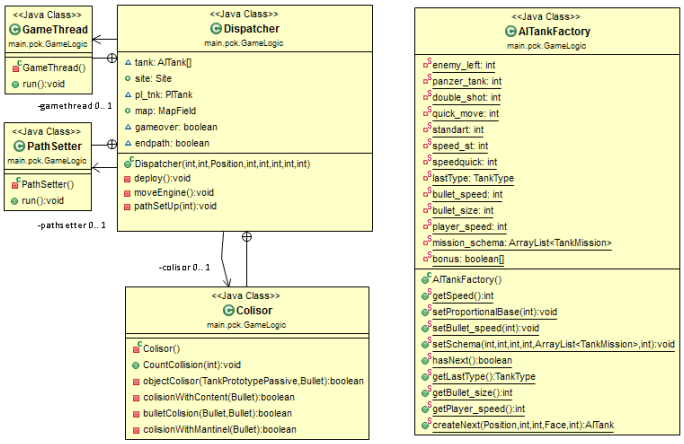
Aby se objekty pohybovaly správnou rychlostí, neudává se jejich rychlost v přímo, ale k poměru k základní jednotce – libovolné konstantě. Ta pokud je zvolena jako 10 a poměr rychlosti nastaven na 2,5 znamená, že při velikosti jednoho pole 20 pixelů se objekt bude pohybovat o 5 pixelů za hrací kolo. Nastavena je až při inicializaci třídy SurfaceViewDeriveted, která pak stanoví velikost jednoho pole. Důležitý bod k zamyšlení je také ten, že při změně velikosti displeje dojde ke změně poměrů ve hře. Pohyb jednotek je proto dán poměrem ke zvolené základní jednotce a počítán podle vzorce:

$$Proportion = (\frac{BaseUnit}{BaseProportion}) * Accutation$$

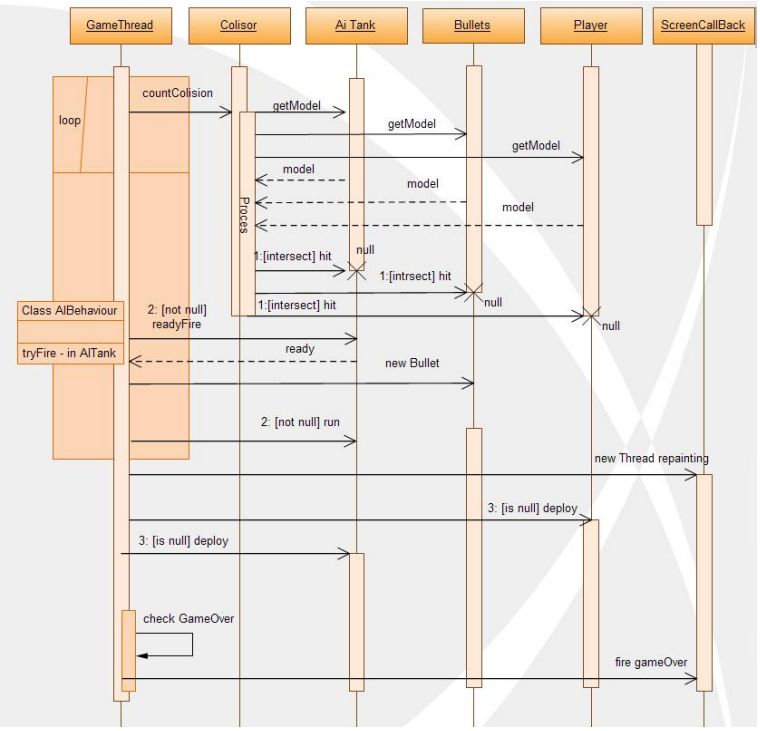
$Proportion$ je výsledná proporce v pixelech. $BaseUnit$ je velikost jednoho pole v pixelech. $Baseproportion$ je libovolně zvolené konstanta (např. 10) Accutation jsou jednotlivé poměry k základní jednotce.



Obrázek 8: Nejvyšší vrstva – třídní diagram



Obrázek 9: Herní vrstva – třídní diagram



Obrázek 10: Běh aplikace – sekvenční diagram

7.3 Hrací pole

Hrací plocha byla zvolena jako dvou rozměrná matice 25x25 polí. Výhodou je snazší umísťování předmětů do hry a hlavně počítání cesty umělé inteligence. Jedna matice slouží jako model pro hledání cesty – Síťový model, druhá pro uchovávání informace co na daném poli je – Mapovací model. Síťový model má pak odkaz na matici, kde se uchovávají detaily co na daném poli je, tedy na Mapovací model. Ten také využívá hlavní dispečer, konkrétně jeho herní vlákno, při detekci kolizí a přímo oba modely obsluhuje a modifikuje data podle aktuálního stavu hry.

Pole pro účely hry a hlavně počítání cesty a umísťování objektů je počítáno v logických jednotkách (25x25) kde y je šířka a x výška. Každý model nese údaj také o své pixelové pozici počítané podle vzorce :

$posPixelX = posLogicY * cubeSize$ (souřadnice jsou oproti sobě obráceně).

Platí tedy že: Logická pozice udává souřadnice matice. Pixelová pozice udává souřadnice obrazovky.

7.4 Jednotlivá kola (prostředí)

Kola ve hře jsou definována XML souborem. Formát XML byl vybrán pro jeho snadnou přenositelnost a množstvím build-in parserů v AOS. Soubor nese údaje o rozmístění jednotlivých předmětů, hráčovy základny a respawn bodů (BASE a PLDEPO). Také obsahuje informace o tom:

- Kolik a jakých typů nepřátel se v kole vyskytne a jejich maximální počet najednou - TSCHEMA, AIMAX - c
- Kolik bude v kole bonusů – jejich typ a kdy se vyskytnou je ale generován náhodně - BONUS
- Jaký bude AI typ mise. Mise může být zadaná k útoku na hráče, nebo na základnu. - MISSION
- Jaké bude zpoždění mezi střelbou. Udává tedy kadenci střelby AI modelů. AIMAX - del

Díky tomu jsou další kola těžší a každé kolo je tak plně nastavitelné. Pořadí jednotlivých typů nepřátel je náhodné, dá se ovlivnit pouze jednotlivý počet typů tanků. Po vyčerpání všech nepřátelských jednotek, se kolo vyhodnotí jako vyhrané a odemyká se další.

Ke generování map byla napsána speciální aplikace generující XML, podle zadaného nákresu. Jde o v Java-SE (Standart Edition) napsanou aplikaci, kde lze pomocí myši v uživatelském rozhraní vysázet kolo. Po té aplikace XML vygeneruje.Výsledný soubor neobsahuje herní schéma, které musí být dopsáno separátně. Zde je ukázka z XML souboru 2

```
<TSCHEMA c="21" pan="6" quick="5" dob="5"/>
<AIMAX c="5" blt_del="13"/>
<MISSION c="PL"/>
<MISSION c="BS"/>
<BASE x="24" y="12"/>
<PLDEPO x="24" y="9"/>
<BONUS c="5"/>
```

Výpis 2: XML schéma

Schéma misí - MISSION - je dáno sekvenčně pro každou novou jednotku. Pokud chybí je implicitně nastavena k útoku na hráče. Stejně tak se dopočítávají nejjednodušší nepřátelé do chybějícího počtu. V našem případě je v TSCHEMA element c nastaven na 21, což značí počet všech nepřátel. Pokud by jsme ostatní elementy nastavili na hodnotu 0, vygenerovali by se pouze jednodušší nepřátelé. Kolo jako takové je pak načítáno do třídy, která slouží jako obal na informace a ty poskytuje zbytku hry viz 8. Hlavní dispečer pak podle stavu hry informace o rozložení mění, a odstraňuje rozstřílené překážky. Všechny tyto informace drží třída Level, která se používá k rychlému vykreslování. Jednotlivé obrázky jsou drženy v paměti jako Hash-tabulka, kde jako klíč je řetězec a hodnotou pixelová pozice. Řetězec se skládá z Logické pozice x a y a např. pozice 3 4 je uložena jako "x3y4". Děje se tak z důvodu rychlého vyhledávání a údaj se používá jako jednoznačný identifikátor.

7.5 Kolize

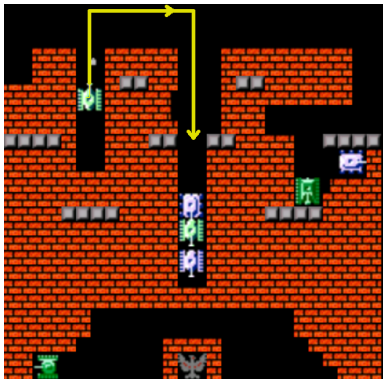
Kolize modelů jsou řešeny pomocí tzv. obalu objektů. Jak jsme již řekli, obal je jednoduchý geometrický útvar, který reprezentuje těleso a u nějž jde snadno spočítat, zda nekoliduje s jiným tělesem. Pomocí build-in Java funkce Rect.intersects se kontroluje vzájemná kolize dvou čtyřhranů. Ty generuje každý model na požádání aktuálně k současné pixelové pozici. U střel je třeba čtyřhran malinko zvětšit kvůli snadnější hře a díky způsobu jakým je střela generována. Při srážce se zdí je využita mapovací třída, která vyhodnocuje, zda na daném poli (v logických jednotkách) je nebo není grafický artefakt, který má být střelou zničen. Je nutné si uvědomit, že tento návrh šetří značnou část výpočtů, namísto porovnávání dvou modelů každý s každým, jak je tomu u tanků a střel. Kolize typu tank-artefakt je také řešena skrz mapovací třídu.

Z kapitoly 6.4. tedy potřebujeme všechny typy kolizí. Jak jejich rozpoznání, tak její zabránění – tanky nesmí opustit vymezené hrací pole, či projíždět zdí. Poslední bod potřebujeme přirozeně na zničení tanku v případě, že objekt zasáhne projektil.

Schéma Android vrstvy. MovementWorker je pouze pomocná třída. Při držení tlačítka pro pohyb hráčem Android 2.3 možnost stisknutí tlačítka nepodporuje. Dispatcher je hlavní třída pro chod hry.

7.6 Modely a AI

Modely jsou ve hře dvojího typu. Jeden čistě logický pro účely herní logiky, druhý jako sprite. Ten tvoří součást Androidní části aplikace. O změnách stavu (upgrade, destrukce,



Obrázek 11: Ukázka postupu AI jednotek – scénář I

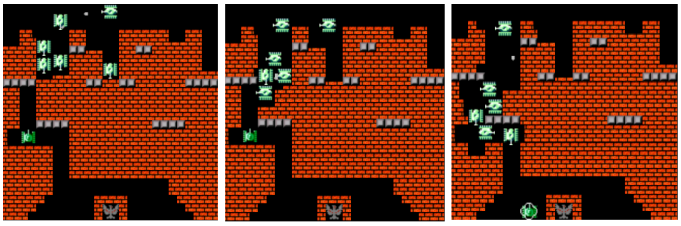
atd.) je herní plátno informováno pomocí callback metod, pozice je předávána referencí. Takže není nutné udávat pozici v každém kole. Modely mají nastavenou svou velikost v logických jednotkách a pixelových. Hráčův tank i AI jednotky dědí ze společné super třídy a je tak možné využívat společných rysů a výhod objektového programování.

U AI jednotek je nutné zpoždění mezi jednotlivou střelbou jinak by programově řízené entity nedávaly šanci lidskému protihráči. Udává ji celé číslo a je každé kolo snižována, až při hodnotě 0 AI jednotka vystřelí. Je nastavitelná skrz XML soubor s jednotlivými koly. Pohyb je umožněn vždy o následující pole libovolně nahoru, dolů nebo do stran. AI jednotky mají cestu přednastavenu jako frontu souřadnic v logických jednotkách. Dispečer vyzvedne z fronty další pozici logickou a převede na pixelovou a model překlopí vnitřní stav na pohybující se. Pak model přesouvá, až dokud nedosáhne pixelové hodnoty.

AI je ve hře reprezentovaná modifikací Dijkstrova algoritmu. Prochází celý graf, a hledá nejrychlejší možnou cestu z množiny všech cest. Umí také procházet a řešit bludiště, má-li jeho řešeni 1 a více možností. Doporučené kola na otestování jsou 1 a 2. Oproti klasické Dijkstrově implemntaci, je časově náročnější - končí ve chvíli, kdy už není další neprozkoumané cesty. Původní řešení bylo rekurzivní, po testech se ale ukázalo, že dochází k přetečení zásobníku. Proto byla vybrána implementace s použitím iterace.

Dispečer má pak možnost každému uzlu nastavit hodnotu (obtížnost), která udává jak těžké je se na daný uzel dostat. Díky tomu AI preferuje cestu, kde nejsou cihly nebo jiné AI jednotky jak ukazují obrázky 11 a 12 . Úkoly se zadávají v XML souboru, jak je psáno v odstavci 7.4. Každému tanku je počítána znovu cesta jednou za cyklus třídou PathSetter.

Největší překážkou byla synchronizace právě vlákna, které počítá cestu soupeřům a herního enginů jak jsme problém nastínili v kapitole 6.1. Každou iteraci PathFinderu – asynchroně běžícího vlákna, je počítána cesta pro jednu AI jednotku. Cesta samotná je ukládána jako řetěz po sobě jdoucích souřadnic, které cestu přesně vytyčují. Je držena v elementech pole PathKeeperů – tříd sloužících jako obal pro cesty. Platí, že ID tanku udává index elementu v poli a příslušné cesty pro dané AI. Před zahájením každého výpočtu je tank zastaven. Cesta je totiž počítána z pozice, na které aktuálně stojí. Po skončení výpočtu, kdy se cesta ukládá je PathKeeper resetován a uložena cesta nová, aktuální vzhledem k pozici hráče.



Obrázek 12: Ukázka postupu AI jednotek – scénář II

V herním engínu se pak zkoumá, zda nebylo AI zastaveno. Po té je nová souřadnice prozkoumána, a pokud je vyhodnocena jako dosažitelná, je pozice odstraněna z fronty a zahájen přesun na novou souřadnici. Je-li nedosažitelná, nebo stojí-li AI, pak je pouze jednotka otočena ve směru dalšího bodu, aby mohla případnou překážku odstranit střelbou.

Výše zmíněné situace se v informatice označují jako kritické sekce a jejich vykonávání nesmí nikdy běžet současně. V případě, že by se tank na dané pole vydal a zároveň by mu byla přepsána původní cesta, mohlo by dojít k zaseknutí jednotky, jelikož nová cesta už by nemusela odpovídat. Stejně tak při zápisu by mohlo zároveň dojít k vyjmutí některého z bodů cesty. Problém je znám také jako producent-konzument a v kódu je řešen elegantní synchronizací oproti celému poli Pathkeeperů. Jako typ kolekce pro uchovávání cesty byl použit linkovaný list. Má vlastnost, díky které je schopen uchovávat si pořadí držených elementů a mazaní kteréhokoliv prvku je časově velmi levné. Stejně tak i přidání nového, takže pro náš účel se jedná o ideální řešení.

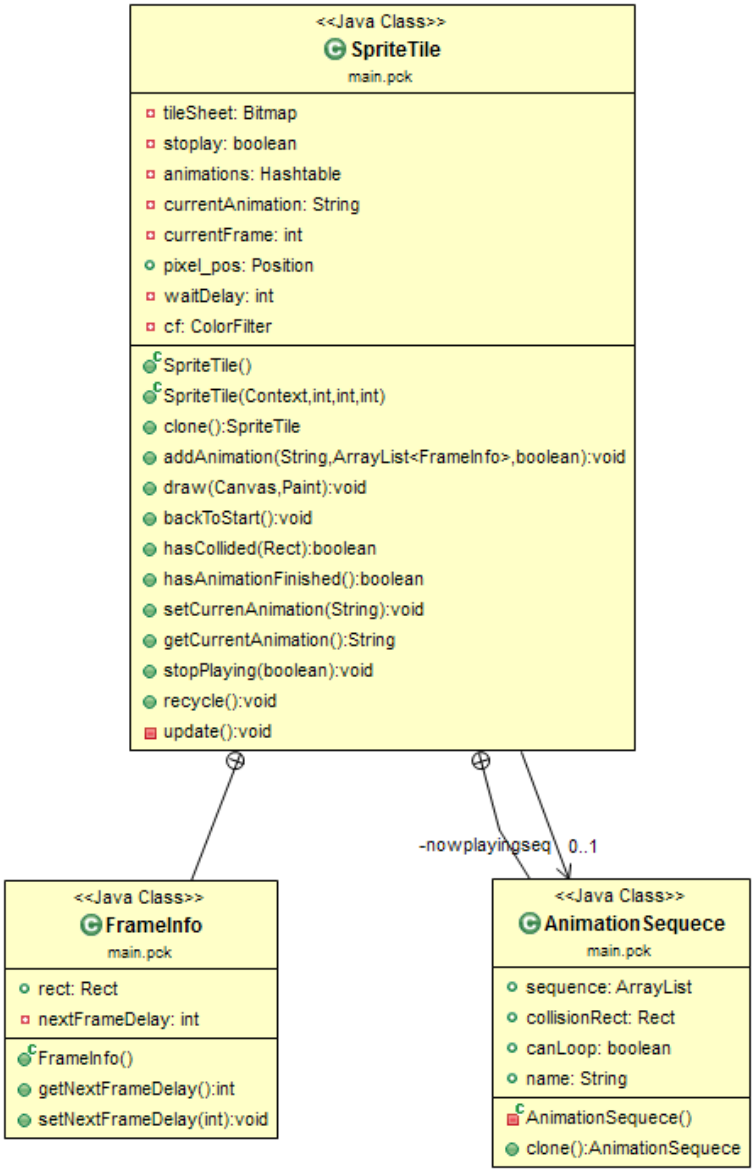
I když z měřených časů z tabulky 1, vyplývá že, na starších zařízeních s nižší verzí androidu běží stejný kód PathFinderu až 10x pomaleji, projeví se to jen sníženou schopností reagovat na změnu polohy hráče. Nepřátelé se tak mohou jevit méně inteligentní. Pokud jedna iterace trvá až 250 milisekund a vlákno je ještě např. na 50 milisekund uspáno, znamená to, že 3 AI jednotkám je propočítána nová cesta přibližně jednou za sekundu. Tank také delší dobu stojí na místě. Jak bylo zmíněno, během výpočtu totiž musí setrvat na stejné logické pozici. Rozdíly jsou dány především JIT kompilerem, který na Androidu 2.1 chybí a ve verzích 3.0 a 4.0 byl navíc dále vylepšen.

7.7 Simulace pohybu a vykreslování

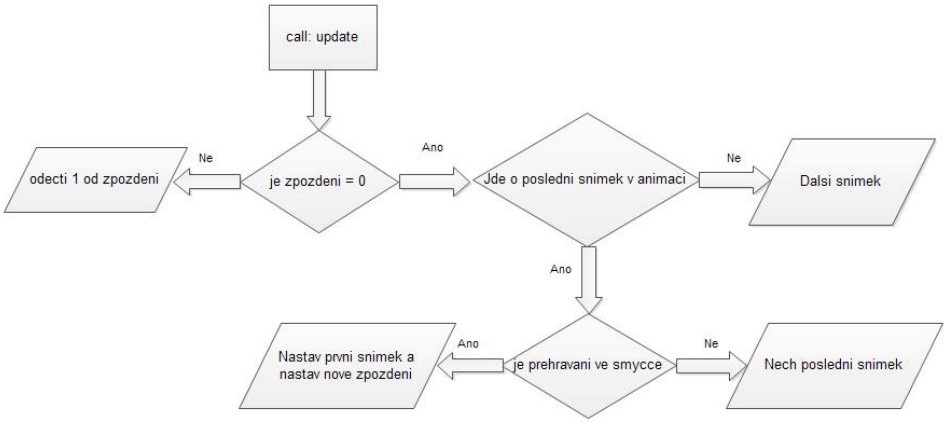
Simulace pohybu – Sprite (Animace) je načítána z .gif souboru. Podmínkou jsou za sebou řazené jednotlivé obrázky/snímky naprosto identické velikosti. Po té jsou z XML souboru extrahována data o pozici jednotlivých snímků na .gif obrázku v pixelech. K uchování této informace slouží zaobalující třída FrameInfo 13. Obsahuje také informaci o prodlevy mezi jednotlivými snímky. Snímky (FrameInfa) jsou pak po skupinkách řazeny do Animací. Ty mají jméno, podle kterého jsou spouštěny. Jedna animace tak ve smyčce přehrává tank směřující doleva, jiná při jízdě vpravo, atd. Můžou obsahovat libovolný počet snímku a je možné pomocí bool proměnné nastavit, zda se mají opakovat ve smyčce, nebo zastavit na posledním snímku. Tento proces popisuje fázový diagram 14. Ke změně zobrazeného

snímku dochází po odpočtu prodlevy proměnné waitDelay. Z kapitoly 6.5 jsem vybral Malířův algoritmus, vzhledem k 2D dimenzi je naprosto postačující a rychlý. Ze stejného důvodu je zde použita animace pomocí Sprite. Tyto třídy a koncept byl převzat z cizího zdroje [14], pro účely hry byl ale lehce modifikován.

Vykreslování samotného prostředí je řešeno pomocí Hash-Tabulek, jak již bylo zmíněno v kapitole 7.4. Tabulky stačí projít a vykreslit jednotlivé bitmapy v pořadí v jakém se mají překrývat. Travnaté plochy jsou tak vykresleny jako poslední, aby překrývaly hráče a ostatní předměty jak tomu má být. Bitmapy pro jednotlivé verze nepřátel jsou drženy v paměti jako šablony. V případě že dojde k vytvoření nové AI entity naklonuje se Sprite ze šablony. Není tak třeba držet v paměti pro každou AI jednotku všechny typy verzí, ale mít jen načtené různé vzory, které jsou po té distribuovány v průběhu hry. AI jednotek může být najednou libovolný počet, pro více než 5 najednou je ale hra pro uživatele značně obtížná.



Obrázek 13: Sprite – Třídní diagram



Obrázek 14: Sprite – Fázový diagram

8 Závěr

Práce shrnula možnosti vývoje herních aplikací a jednu implementovala. Architektura a provedení jednotlivých částí této vybrané hry ukázala jakým způsobem lze samotnou hru implentovat.

Aplikace byla v první části vyvíjena na mobilním zařízení ZTE s AOS 2.1. V druhé fázi na Sony XPERIA mk16-i kde byl chod aplikace znatelně rychlejší. S testováním a návrhem kol pomáhal externí tester, jehož úkolem bylo generování a úprava XML souboru, a na tuto práci musel být zaškolen. Díky snadné přenesitelnosti XML souborů, bylo jejich generování platformě nezávislé.

Z pohledu nároků na hardware není hra náročná, což bylo také zamýšleno. Postačí zařízení s verzí Androidu 2.1 nebo výší. Na těchto zařízeních, ale běží herní vlákna pomaleji z důvodu absence JIT kompileru viz tabulka 1 . Ukazuje jednotlivé průměrné časy pro tři samostatná vlákna. Včetně průměrné odchylky, která udává o kolik jednotlivé časy oscilují. Jak bylo zmíněno v kapitole 7.6 na technicky slabším přístroji je inteligence a reakce nepřátel mírně utlumená. Uživatel by však měl toto omezení pocítit zcela minimálně. Verze 2.1 byla zvolena z důvodu její stále velké rozšířenosti v ČR a také z důvodu testovacích zařízení. Aplikace běží bez problému na kterémkoliv zařízení se zmíněným operačním systémem.

Operace		Sony-Ericsson Xperia MK-16i Android 4.1	ZTE 945 Android 2.1 (Updated)
Iterace PathFinderu	Průměr Prům. odchylka	19,0ms 9,1ms	208,5ms 17,7ms
Překreslení obrazovky	Průměr Prům. odchylka	22,4ms 4,6ms	29,3ms 9,7ms
Herní kolo	Průměr Prům. odchylka	3,9ms 1,8ms	25,9ms 10,6ms

Tabulka 1: Naměřené časy

Hrubý odhad času potřebného na kompletní vývoj a odlazení aplikace byl stanoven na cca. 150-200 hodin. Ze softwarové stránky, je aplikace nenáročná. V práci nebyla použita žádná externí knihovna a všechny části kódu jsou vytvořené autorem, až na koncept a implementaci Sprite animací (viz. kapitola 7.7). Velkou výzvou bylo použití a modifikace Dijkstrova algoritmu - do poslední chvíle nebylo jisté, jestli při základní implementaci zařízení zátěž zvládne. Kód byl také od svého prvního releasu značně upraven a zjednodušen, takže nároky na paměť jsou rovněž podstatně menší.

Do budoucna, jako možné rozšíření, by bylo nasnadě použít například A star algoritmus, který by dával hráči možnost sledovat jak se nepřátelé reálně učí jakou strategii zvolit. Dalo by se tak docílit s použitím genetických algoritmů, kdy nejlepší jedinec předává zkušenosti potomkům. Dalším vylepšením by pak mohla být verze pro více hráčů skrze wi-fi, úprava vyhledávacího algoritmu na více vláknový (např. 2), nebo hru implementovat ve 3D. Byla zde také myšlenka engine hry předělat ve framework, ale z důvodů

časové náročnosti, a hlubokých zásahů do konceptů celého systému, nebyla nakonec tato myšlenka – stejně jako 3D verze – realizována.

Martin Škorec

9 Reference

[1] Email Marketing Reports, *Smartphone statistics and market share* [Online] http://www.email-marketing-reports.com/wireless-mobile/smartphone-statistics.htm

[2] App Brain, *Number of available Android applications*, [Online 2012] http://www.appbrain.com/stats/number-of-android-apps

[3] Sinnathamby Mark, *Stack based vs Register based Virtual Machine Architecture.* [Online 2012] http://www.codeproject.com/Articles/461052/Stack-based-vs-Register-based-Virtual-Machine-Arch

[4] Rovio©, *1 BILLION Angry Birds downloads!* [Online 2012] http://www.rovio.com/en/news/blog/162/1-billion-angry-birds-downloads

[5] Lamport, Leslie, *Writing real time games for android.* ,Vortrag Google IO, 2009.

[6] OpenGL®, *OpenGL® ES 2.0 Programming Guide - Book Website*, [Online] http://opengles-book.com/

[7] Microsoft©, *Differences in game development between the phone and the desktop* [Online] http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj662930%28v=vs.105%29.aspx

[8] Apple©, *Distribute your apps on the Mac App Store* [Online] https://developer.apple.com/devcenter/mac/checklist/

[9] Microsoft©, *Submitting your app* [Online] http://msdn.microsoft.com/en-us/library/windows/apps/br230835.aspx

[10] Android©, *Signing Your Applications* [Online] http://developer.android.com/tools/publishing/app-signing.html

[11] Android©, *Jelly Bean* [Online] http://developer.android.com/about/index.html

[12] Hazelden,Alan, *Physics and Collision Detection* [Online] http://www.draknek.org/physics/wgd-talk.pdf

[13] Pavlovský, Jan, *Virtuální světy na PDA.* České vysoké učení technické v Praze, 2010

[14] Guerrero Maximo, *Android Game Development – Part 1 GameLoop & Sprites* http://warriormill.com/2009/10/adroid-game-development-part-1-gameloop-sprites/

[15] Martin John Baker, *Physics - Impulse* [Online 2008] http://www.euclideanspace.com/physics/dynamics/collision/impulse/index.htm

[16] GITTA, *Dijkstra Algorithm: Short terms and Pseudocode* [Online 2008] http://www.gitta.info/Accessibiliti/en/html/Dijkstra_learningObject1.html